Distributed Systems

COMP90015 2019 Semester 1
Tutorial 8

Things to cover in Tutorial 8

Security Questions

SSL Explanation! Why do we need it?

Code Demonstration : SSL Demonstration

Security Questions

1. List and briefly explain some worst case assumptions when designing a secure system.
2. Define encryption and describe the two main types of keys used by encryption algorithms.
3. Discuss the three major roles that encryption plays in the implementation of secure systems.
4. Explain how digital signatures work.

Security Questions

5. How does Alice send a secret message to Bob if they both share a secret key?
6. How can Alice authenticate and communicate secretly with Bob assuming there is an authentication server that knows Alice's and Bob's secret keys?
7. Assuming that Bob has a public/private key pair, how can Alice and Bob establish a shared key to communicate secretly using a Key Distribution Service?

1. List and briefly explain some worst case assumptions when designing a secure system.

- **Networks are insecure.**
  - Messages can be looked at, copied, modified and retransmitted.
  - Attackers can obtain information that they should not and can pretend to be a legitimate party.
- **The source code is known to the attacker.**
  - Knowing the source code can help the attacker discover vulnerabilities.
- **Interfaces are exposed**
  - Communication interfaces are necessarily open to allow clients to access them.
  - Attackers can send messages to any interface.
- **The attacker has unlimited computing resources.**
  - Assume that attackers will have access to the largest and most powerful computers projected in the lifetime of a system.

2. Define encryption and describe the two main types of keys used by encryption algorithms.

- ***Encryption***
  - process of encoding a message in such a way as to hide its contents.
- ***Shared secret keys*** (symmetric)
  - Sender and recipient share knowledge of the key and it must not be revealed to anyone else.
- ***Public/private key pairs*** (asymmetric)
  - The sender uses a public key to encrypt the message.
  - The recipient uses a corresponding private key to decrypt the message.
  - Only the recipient can decrypt the message, because they have the private key.
  - Typically require 100 to 1000 times as much processing power as secret-key algorithms.

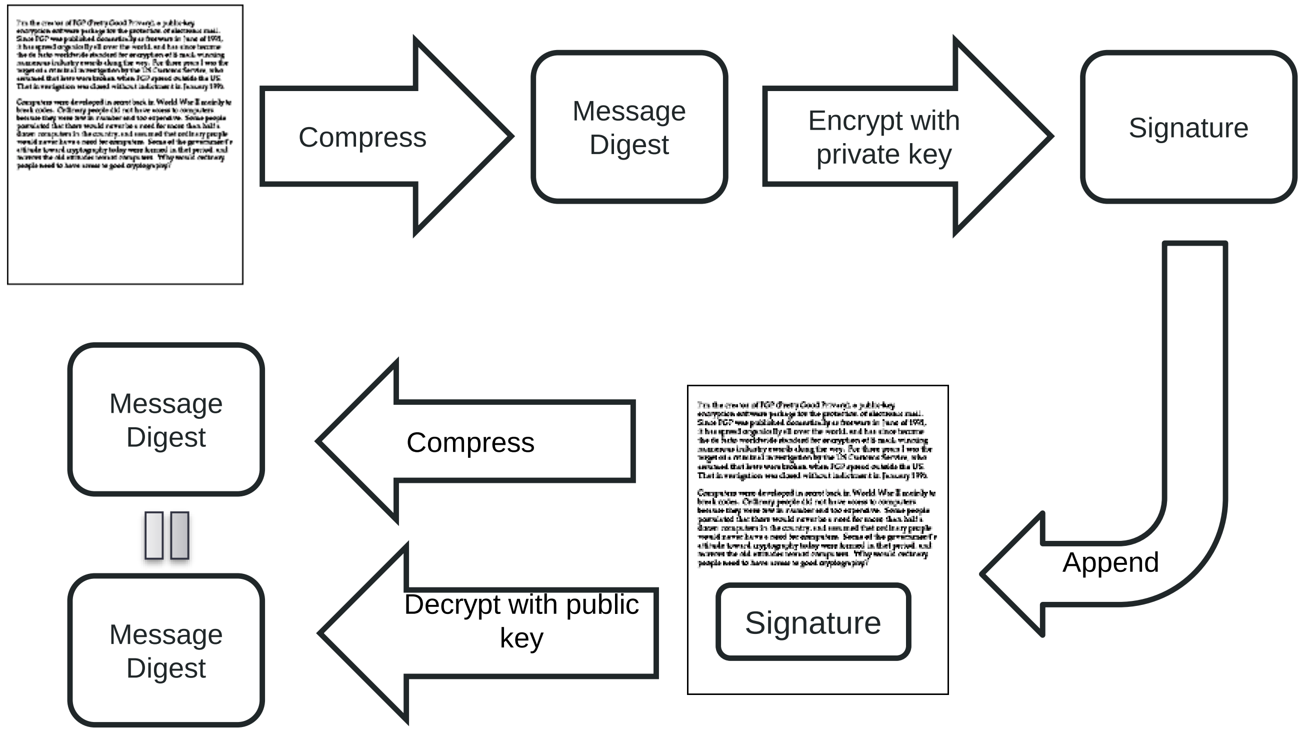
3. Discuss the three major roles that encryption plays in the implementation of secure systems.

- **Secrecy and integrity**
  - Messages encrypted with a particular key can only be decrypted by a recipient who knows the corresponding decryption key => Secrecy.
  - Integrity can be maintained if some redundant information such as a checksum is included and checked in the encrypted message.
- **Authentication**
  - If keys are held in private, a successful decryption authenticates the decrypted message as coming from a particular sender.
- **Digital signatures**
  - Verify to a third party that a message or a document is an unaltered copy of one produced by the signer.
  - It is a "stamp" Bob places on the data which is unique to Bob, and is very difficult to forge.
  - It also assures that any changes made to the data that has been signed cannot go undetected.

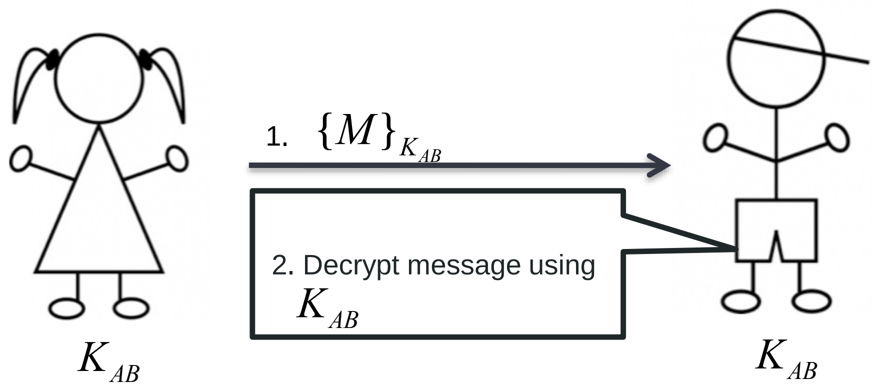
4. Explain how digital signatures work.

1. To sign a document, Bob first ‘compresses’ the message into just a few lines. This is called a *digest*.
2. Bob then encrypts the message digest with his private key. The result is the digital signature.
3. Bob appends the digital signature to the document. All of the data that was ‘compressed’ into the digest has been signed.
4. Bob sends the document to Alice.
5. Alice decrypts the signature (using Bob's public key) changing it back into a message digest.
6. Alice ‘compresses’ the document data into a message digest. If the message digest is the same as the message digest created when the signature was decrypted, then Alice knows that the signed data has not been changed and that Bob signed the document (because only Bob has his private key)

4. Explain how digital signatures work.

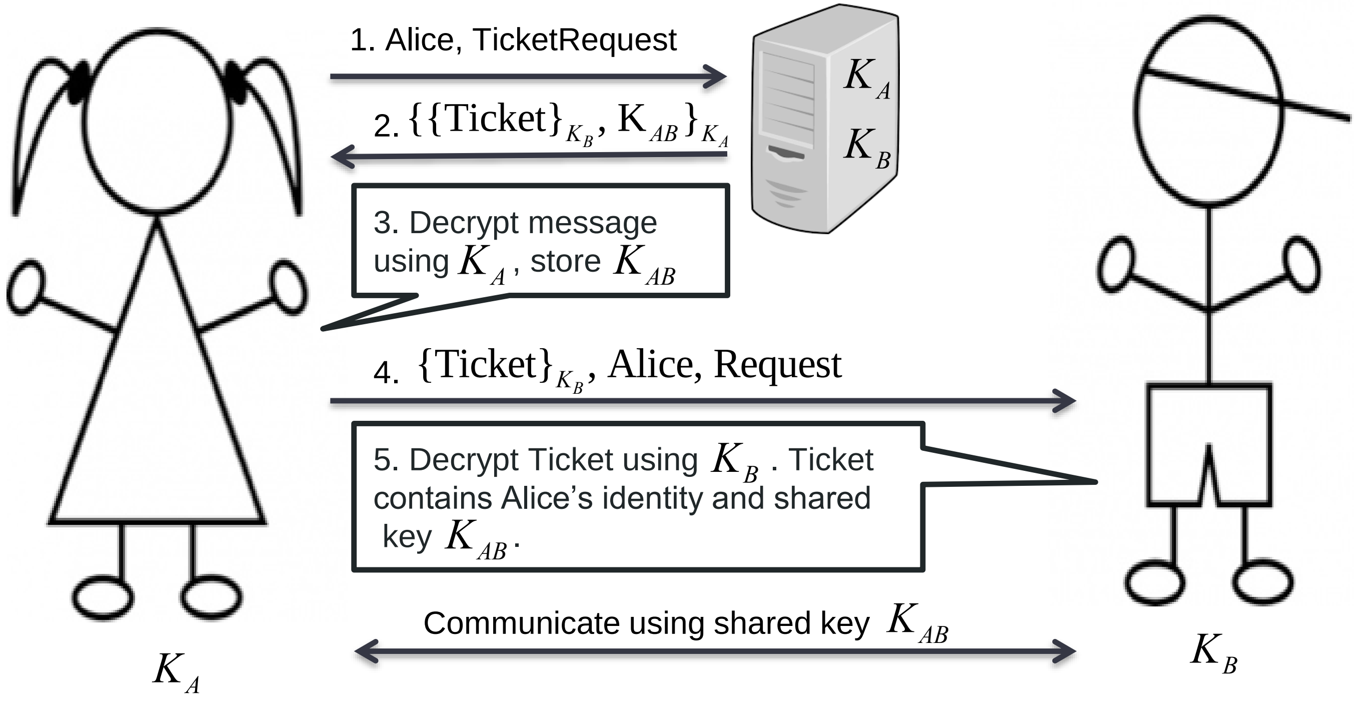


5. How does Alice send a secret message to Bob if they both share a secret key?

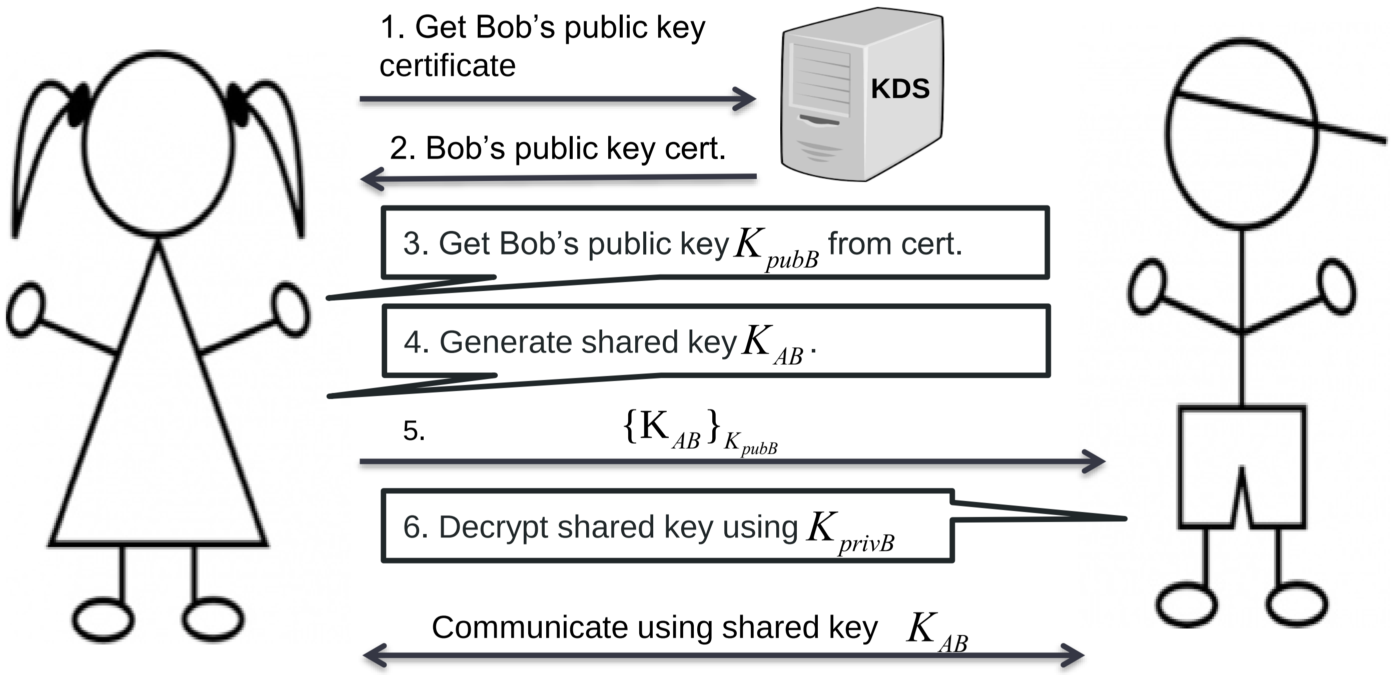


1. Alice uses K_AB and an agreed encryption function *E*(*K_AB, M*) to encrypt and send the message to Bob.
2. Bob decrypts the encrypted message using the corresponding decryption function *D*(*K_AB*, *M*).

6. How can Alice authenticate and communicate secretly with Bob assuming there is an authentication server that knows Alice's and Bob's secret keys?


1. Alice, TicketRequest
K_A
K_B
2. $\{\{\text{Ticket}\}_{K_B}, K_{AB}\}_{K_A}$
3. Decrypt message using K_A, store K_{AB}
4. $\{\text{Ticket}\}_{K_B}$, Alice, Request
5. Decrypt Ticket using K_B. Ticket contains Alice's identity and shared key K_{AB}.
Communicate using shared key K_{AB}
K_A
K_B

7. Assuming that Bob has a public/private key pair, how can Alice and Bob establish a shared key to communicate secretly using a Key Distribution Service?


1. Get Bob's public key certificate
KDS
2. Bob's public key cert.
3. Get Bob's public key K_{pubB} from cert.
4. Generate shared key K_{AB}.
5. $\{K_{AB}\}_{K_{pubB}}$
6. Decrypt shared key using K_{privB}
Communicate using shared key K_{AB}

Digital Certificate

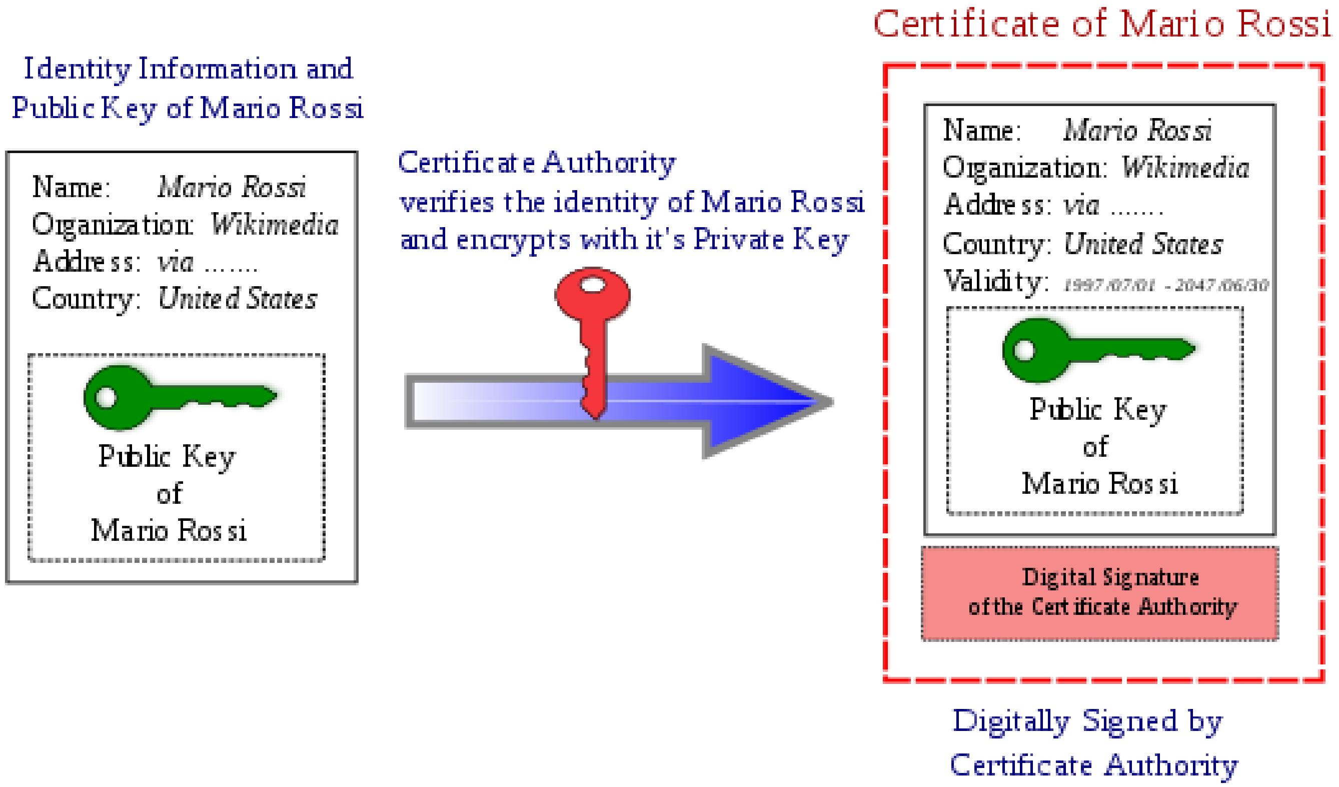
A document containing a statement that is signed by a **Certificate Authority**.

Certificates are used to associate a public key with an identity. Ensuring communicating parties that Alice is really Alice or that Bob is really Bob.

Certificate example

- Let the Certificate Authority be **Sara** and the communicating parties be **Alice** and a **bank**.
- Alice wants to be connected to the right bank. Alice does not know if the bank's public key really belongs to the bank but Alice trusts that Sara knows.
- If Sara knows the bank then she can guarantee that the bank's public key is correct, through a certificate.
- Sara signs the bank's certificate with her own private key. This means that everyone else can validate the certificate using Sara's public key, ensuring that it is genuine.

Public Key Certificate



SSL

Secure Socket Layer is a protocol that is intended to provide a flexible means for clients and server to communicate using a secure channel.

It prevents eavesdropping, tampering and message forgery.

To use SSL, you need a Certificate that will identify who you are.

How does it work?

SSL Protocol

1. Agree on the cipher and hash functions that both client and server support.
2. Server authenticates by sending a certificate.
   The certificate contains: server name, trusted certificate authority, server's public key.
3. The client confirms the validity of the certificate.
4. Shared session key for the secure connection is generated.
   Client can encrypt a random number with the server's public key and send it to the server. Only the server can decrypt the random number with its private key. The agreed on random number is used to generate a unique session key.

After the steps above, secured connection begins. Data is encrypted and decrypted with the session key until the connection closes.

SSL and Java

Java Secure Socket Extension (JSSE) provides a set of packages which enable secure Internet communications.

It implements a Java technology version of the SSL/TLS protocols.

- `SSLServerSocketFactory (Server)`
- `SSLServerSocket (Server)`
- `SSLSocket (Client and Server)`
- `SSLSocketFactory (Client)`

SSL and Java

- Build server certificate
  - Keytool (part of JSE).
  - Allows users to administer their own public/private key pairs and associated certificates.
  - Create keystore, generate a self-signed certificate.
  - `keytool -alias` ***`<aliasName>`*** `-genkey -keyalg` **`RSA`** `-keystore` ***`<path/to/keystore/keystoreName>`*** `-storepass` **`<password>`** `-validity` **`<days>`** `-keysize` **`<2048>`**
- Define keystore location in server:
  - `javax.net.ssl.keyStore=`***`<path/to/keystore/SERVERkeystoreName>`***
  - `javax.net.ssl.keyStorePassword=`***`<password>`***
- Define trusted certificate in client:
  - `javax.net.ssl.trustStore=`***`<path/to/keystore/CLIENTkeystoreName>`***

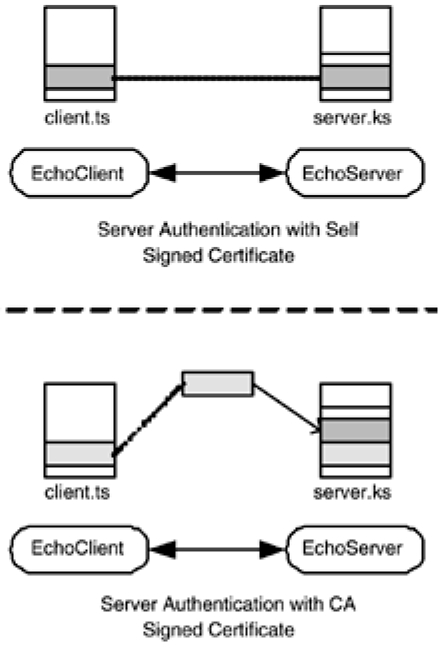
SSL and Java

- The server will export their certificate to the client
  - `keytool -alias` ***`<aliasName>`*** `-export -file` ***`<certificateName.crt>`*** `-keystore` ***`<path/to/keystore/SERVERkeystoreName>`***
- The client will then import the certificate
  - `keytool -keystore` ***`<path/to/keystore/CLIENTkeystoreName>`*** `-import -file` ***`<certificateName.crt>`***

Generate a CSR:

- `keytool -certreq -alias` **`<aliasName>`** `-file` **`<certificateName.csr>`** `-keystore` **`<path/to/keystore/keystoreName>`**

Diagram to show the interaction


client.ts
server.ks
EchoClient
EchoServer
Server Authentication with Self Signed Certificate
client.ts
server.ks
EchoClient
EchoServer
Server Authentication with CA Signed Certificate

Code Demo!